Scalable Multi-Agent Maze Traversal With Local Communication

Julian Rau[1][0009−0006−0913−6555], Jahir Argote-Gerald[2][0009−0003−1492−1329], Grace McFassel[1][0000−0002−8568−1313], Genki Miyauchi[3][0000−0002−3349−6765], Paul Trodden[2][0000−0002−8787−7432], and Roderich Groß[1,2][0000−0003−1826−1375]

[1] Department of Computer Science, Technical University of Darmstadt, Germany
{julian.rau, grace.mcfassel, roderich.gross}@tu-darmstadt.de
[2] School of Electrical and Electronic Engineering, The University of Sheffield, UK
{jaargotegerald1, p.trodden}@sheffield.ac.uk
[3] Bristol Robotics Laboratory, University of Bristol, Bristol, UK
g.miyauchi@bristol.ac.uk

Abstract. Cave networks, pipe systems, and similar maze-like environments pose significant challenges for multi-agent navigation in unknown settings with limited communication. We propose a distributed algorithm that enables agents to collectively traverse an unknown, possibly cyclic graph. Agents enter sequentially at a designated start node and are tasked to localize and reach an undisclosed goal while avoiding collisions. They coordinate via local communication using leader–follower relationships and leader switching. At any moment in time, exploration is performed by only one of the agents, which runs a single-agent maze solver. We prove that the algorithm is complete, that its makespan is asymptotically equivalent (in the number of agents) to that of an optimal full-knowledge strategy, and derive its time and space complexity. Simulations with up to 625 agents show a decreasing average sum-of-fuels as the number of agents increases and demonstrate that the proposed approach outperforms a naïve baseline in which all agents independently execute the single-agent solver.

Keywords: Multi-Agent Maze Traversal · Multi-Agent Systems and Distributed Robotics · Algorithmic Completeness and Complexity

1 Introduction

Many real-world scenarios require multiple agents to navigate efficiently through complex, maze-like environments such as networks of pipes [18, 27] or caves [15, 13, 19]. Coordinating such agents poses unique challenges, including the uncertainty of environments, communication restrictions, and inter-agent collisions.

Environments are often modeled as graphs. If the graph is unknown a priori, its nodes and edges are locally revealed as agents traverse it. For single-agent systems, classical methods such as breadth-first search (BFS) and Trémaux's algorithm can be employed to search the graph [3, 5, 8]. For multi-agent system,

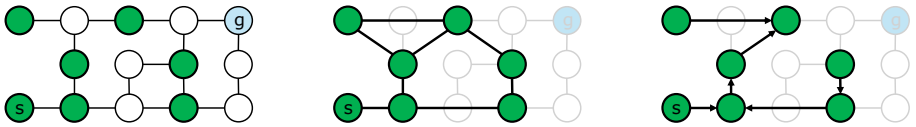
inter-agent collisions and communication have to be considered. Some studies consider agents that explore (tree or) graph environments while coordinating through beacons they place at nodes [2, 4, 9, 16] or through local or global communication [7, 12]. Except for [12], these studies provide theoretical guarantees such as completeness and exploration-time bounds but do not all consider inter-agent collisions [2, 7, 9]. Moreover, their focus is on exhaustive exploration rather than reaching specific goal nodes. The search algorithm proposed in [6] aborts in the event undisclosed goal nodes are discovered. However, it is centralized.

A related problem is Multi-Agent Path Finding (MAPF), which involves identifying collision-free paths of multiple agents to designated goal nodes in a known graph environment [22, 24–26]. In classical MAPF settings, agents are assumed to have full knowledge of the graph, their own start and goal nodes, and those of other agents. Planning is typically conducted in a centralized manner [22, 24, 26]. Some MAPF variants relax these assumptions by requiring agents to discover parts of their environment or plan their paths in a distributed manner [10, 17, 20, 23], but do not permit fully unknown environments.

Multi-Agent Maze Traversal (MAMT) [11, 1] is a problem that combines multiple of the aforementioned challenges. Specifically, it concerns collective, collision-free navigation of arbitrary large group of agents (i) through a unknown graph environment that is only gradually revealed through local sensing as the agents move, and (ii) towards a common, undisclosed goal node. In [11], a distributed approach for MAMT is presented but assumes global communication among agents. In [1], we proposed a distributed approach requiring only local communication. It employs a leader-switching strategy by which one agent (the *head*) executes a single-agent maze solver to search for the goal, while all other agents choose neighboring agents to follow, thereby directly or indirectly following the head. If movement were to result in a collision, the head instead passes its role to the respective agent. Both approaches are limited to acyclic graph environments (i.e., trees) and offer no theoretical guarantees [11, 1], leaving decentralized, locally coordinated MAMT for general graphs an open problem.

The contributions of this work are as follows: (i) We propose a new MAMT algorithm, based on the one presented in [1], to address the MAMT problem for general (i.e. possibly cyclic) graphs. The algorithm manages competition between agents and maintains connected communication within the swarm. It employs multiple messaging rounds per time step and diverse message types. (ii) We formally prove completeness and that the makespan is asymptotically equivalent (with respect to the number of agents) to the one achieved by an optimal strategy assuming full knowledge. (iii) We derive the algorithm's time and space complexity. (iv) We validate the algorithm in simulation with up to 625 agents, showing a decrease in average sum of fuel with growing numbers of agents, and that the algorithm outperforms a naïve baseline (where all agents run the single-agent maze solver) and approaches the performance of the full-knowledge strategy.

Sections 2 and 3 define the MAMT problem and propose and analyze an algorithmic solution. Sections 4 and 5 present simulation results and conclusions.



(a) (b) (c)

Fig. 1. (a) The maze is represented as a graph with start node s, goal node g (blue). Agents (green) are tasked with moving from s to g without prior knowledge of the environment. (b) Agents can only communicate with other agents that are either in an adjacent node or two nodes away with an empty node in between. (c) All agents but one choose a leader to follow, resulting in a leader-follower relation.

2 Problem Formulation

The maze is represented as a connected, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. An example graph is shown in Fig. 1a. Two nodes in this graph, $s, g \in \mathcal{V}$, $s \neq g$, represent the start and the goal (blue), respectively. Node s is assumed to be a leaf in $\mathcal{G}$[4].

A set of n agents, $\mathcal{A} = \{1, \ldots, n\}$, is considered, where $v_i[k] \in \mathcal{V}$ denotes the node that agent $i \in \mathcal{A}$ resides on at time $k \in \mathbb{N}_0$. We define time step $k+1$ as the interval $[k, k+1)$. For simplicity, we omit k when it is clear from the context. At time $k = 0$, all agents are at the start (i.e., $\forall i : v_i[0] = s$) and one agent is considered *active* whereas all others are *inactive*. Active agents remain active indefinitely. At the precise time an active agent leaves the start node, an inactive agent present at the same node (if any) becomes active. Activation repeats until the last agent becomes active and occurs in a fixed, predetermined total order $\leq_{\mathcal{A}}$ on $\mathcal{A}$. Let $\mathcal{A}^{\text{active}}[k] \subseteq \mathcal{A}$ denote the set of active agents at time k.

Let $\mathcal{N}_i[k] = \{u \in \mathcal{V} \mid \{v_i, u\} \in \mathcal{E}\}$ be the set of adjacent nodes of agent i. In time step $k+1$, agent i can either *move* to a node in $\mathcal{N}_i[k]$ or remain in its current node $v_i[k]$. A node $v \in \mathcal{V} \setminus \{g\}$ is occupied at time k, if $\exists i \in \mathcal{A}^{\text{active}}[k] : v_i[k] = v$. A *vertex conflict* [25] occurs at time k if two active agents occupy the same non-goal node, that is, $\exists i, j \in \mathcal{A}^{\text{active}}[k], i \neq j : v_i[k] = v_j[k] \neq g$. A *following conflict* [25] occurs if an active agent moves to a non-goal node in the same time step that another active agent left it, formally, $\exists i, j \in \mathcal{A}^{\text{active}}[k]$, $i \neq j : v_i[k] = v_j[k+1] \neq g$. Both vertex and following conflicts are prohibited. Note that an agent i that activates in timestep $k+1$ is not yet active at k (i.e., $i \notin \mathcal{A}^{\text{active}}[k]$) and therefore does not induce a following conflict with the active agent $j \in \mathcal{A}^{\text{active}}[k]$ that left the start node in time step $k+1$.

At time k, agent i knows[5] (adapted from [1]):

[4] As graphs representing real-world environments are artificially created, we assume they can be constructed accordingly.

[5] For ease of notation, we write $\mathcal{N}_i$ as if agent i had access to the global labels of neighboring nodes, $\mathcal{V}$. However, agent i can only distinguish these nodes, requiring labels that are consistent from agent i's local perspective.

- its unique *ID* i
- whether the node $v_i[k]$ it currently resides on is the goal node, $v_i[k] = g$
- if g is an adjacent node, that is, if $g \in \mathcal{N}_i[k]$, and if so, which edge leads to g
- the subset of adjacent nodes that are currently occupied, $\mathcal{N}_i^{\text{occupied}}[k] = \left\{u \in \mathcal{N}_i[k] \setminus \{g\} \mid \exists a \in \mathcal{A}^{\text{active}}[k] \colon v_a[k] = u\right\}$. The subset of unoccupied adjacent nodes can be obtained by $\mathcal{N}_i^{\text{unoccupied}} = \mathcal{N}_i \setminus \mathcal{N}_i^{\text{occupied}}$.

Messages among agents are limited to a two-hop distance and can only pass unoccupied nodes[6] (example shown in Fig. 1b), resulting in the undirected communication graph $\mathcal{G}^{\text{com}} = (\mathcal{V}^{\text{com}}, \mathcal{E}^{\text{com}})$, where $\mathcal{V}^{\text{com}} = \mathcal{A}^{\text{active}}$ and [1][7]

$$\mathcal{E}^{\text{com}} = \Big\{ \{i,j\} \subseteq \mathcal{A}^{\text{active}} \mid i \neq j \wedge \Big(v_i = v_j \vee \{v_i, v_j\} \in \mathcal{E} \vee \Big(\mathcal{N}_i^{\text{unoccupied}} \cap \mathcal{N}_j^{\text{unoccupied}} \neq \emptyset \Big) \Big) \Big\}.$$

Consequently, agent i can communicate with agents in its communication range $\mathcal{C}_i = \{j \in \mathcal{V}^{\text{com}} \setminus \{i\} \mid \{i,j\} \in \mathcal{E}^{\text{com}}\}$ [1]. Agents can send a message via directed cast, that is, via a specified edge. Formally, when agent i sends a message via directed cast along edge $\{v_i, u\} \in \mathcal{E}$, $u \in \mathcal{N}_i$, this message reaches all agents that currently reside in any node of the set

$$\mathcal{U}_i[u] \coloneqq \left\{ w \in \mathcal{V} \mid w = u \vee \left(u \in \mathcal{N}_i^{\text{unoccupied}} \wedge \{u, w\} \in \mathcal{E} \right) \right\}.$$

Agents can also send a message via (local) broadcast, that is, along all outgoing edges of their current node [1]. A broadcast message sent by agent i reaches all agents currently residing in any node of set $\mathcal{B}_i \coloneqq \bigcup_{u \in \mathcal{N}_i} \mathcal{U}_i[u]$. A message contains the information whether it was sent via directed cast or broadcast. Agents can send and receive messages at any time.

Communication is situated, allowing agent i to determine the node(s) from which a message reached the agent. An agent i that receives a directed cast message from an agent $j \neq i$ via a shared, unoccupied, adjacent node u can determine that this directed cast came from u. If j is adjacent to i and sends a directed cast message via the edge $\{v_i, v_j\} \in \mathcal{E}$ or $v_i = v_j$, i knows that this message came from v_j. An agent i that receives a broadcast message that was sent by an agent $j \neq i$, gathers the nodes from which the broadcast reached i in the set $\texttt{NodesTowards}_i(j)$ (adapted from [1]). Formally,

$$\texttt{NodesTowards}_i(j) = \begin{cases} \{v_j\}, & \text{if } v_j = v_i \\ \{v_j\} \cup (\mathcal{N}_i^{\text{unoccupied}} \cap \mathcal{N}_j^{\text{unoccupied}}), & \text{if } \{v_i, v_j\} \in \mathcal{E} \\ \mathcal{N}_i^{\text{unoccupied}} \cap \mathcal{N}_j^{\text{unoccupied}}, & \text{otherwise.} \end{cases}$$

All agents are tasked to reach the goal g. The following two evaluation criteria are used to measure the performance [25, 1]: (i) *Makespan*: The overall time it took for all agents to reach the goal, formally $\min\{k \in \mathbb{N}_0 \mid \forall a \in \mathcal{A} : v_a[k] = g\}$; (ii) *Average sum-of-fuels*: The mean number of edges traversed to reach the goal, formally $1/n \sum_{i=1}^{n} \pi_i$, where π_i is the number of edges traversed by agent i.

[6] This models constrained local communication, such as line-of-sight communication.
[7] The expression $v_i = v_j$ in $\mathcal{E}^{\text{com}}$ covers the case where two active agents reside in g.

Algorithm 1: Multi-Agent Maze Traversal Algorithm

$v \leftarrow s$, $v^{prev} \leftarrow s$, $L \leftarrow$ `nil`
// Synchronized sensing and messaging step
Sense adjacent nodes
Broadcast and receive *status messages* from neighbor agents, Update $\mathcal{C}$
if status message *received from agent* l **then**
  $L \leftarrow l$
while $v \neq g$ **do**
  $D \leftarrow v$, *competing* $\leftarrow$ `false`
  // Decision making step
  if $L = nil$ **then**
    // Head decision step
    if $g \in \mathcal{N}$ **then**
      $D \leftarrow g$
    else
      $D^{\text{solver}} \leftarrow$ `SingleAgentMazeSolver`()
      if $D^{solver} \in \mathcal{N}^{occupied}$ **then**
        $L \leftarrow$ `Selector`$(\{a \in \mathcal{C} \mid D^{\text{solver}} \in$ `NodesTowards`$(a)\})$
      else if $D^{solver} \neq D$ **then**
        Directed cast along $\{v, D^{\text{solver}}\}$
        $D \leftarrow D^{\text{solver}}$
    Broadcast *head message*
  else
    $D, L \leftarrow$ `ResolveConflict`()
  // Synchronized movement step
  move to D
  $v^{prev} \leftarrow v$, $v \leftarrow D$
  // Synchronized sensing and messaging step
  Sense adjacent nodes
  if $v^{prev} \neq v$ **then**
    Directed cast along $\{v, v^{prev}\}$
  Broadcast *status messages*
  Receive directed casts (optional) and *status messages*
  Update v_L^{prev}, $\mathcal{C}$

3 Algorithm Design

We extend the approach in [1] to cyclic graphs while preserving its core algorithmic idea. In each time step (until reaching a node adjacent to the goal), one active agent (referred to as the *head*) employs a single-agent maze solver to explore the maze. All other active agents choose an active agent, their *leader*, to follow such that the resulting leader-follower-graph is a tree; hence, they directly or indirectly follow the head (Fig. 1c). All agents are assumed to possess any capabilities that are needed for the underlying single-agent maze solver in addition to the requirements discussed in Section 3.2.

Algorithm 2: ResolveConflict (as performed by agent i)

Output: The node to move to D and the updated leader L

if $\{h \in \mathcal{C} \mid L_h = \texttt{nil} \wedge v_h \neq g\} \neq \emptyset$ **then**
  $h \leftarrow \texttt{Selector}(\{h \in \mathcal{C} \mid L_h = \texttt{nil} \wedge v_h \neq g\})$
  `// Wait for head decision step to finish`
  Wait until directed cast (optional) and *head message* from h received
  if head message *contains* $L_h = i$ **then**
    Broadcast *competing message*
    return $v, \texttt{nil}$
  else if *Directed cast from* h *received* $\vee$ $(\texttt{NodesTowards}(h) \cap \mathcal{N}^{occupied} \neq \emptyset)$ **then**
    Broadcast *competing message*
    return v, h
if $g \in \mathcal{N}$ **then**
  Broadcast *competing message*
  return g, L
if $\texttt{NodesTowards}(L) \cap \mathcal{N}^{occupied} \neq \emptyset$ **then**
  Broadcast *competing message*
  return v, L
competing $\leftarrow$ `true` `// Compete for` v_L^{prev}
Broadcast *competing message*
Wait until *competing messages* from all agents $a \in \mathcal{C}$ received
$\mathcal{R} \leftarrow \{a \in \mathcal{C} \mid v_a \neq g \wedge competing_a \wedge L_a \in \mathcal{C} \wedge v_{L_a}^{prev} = v_L^{prev}\}$
if $\texttt{Selector}(\mathcal{R} \cup \{i\}) = i$ **then**
  return v_L^{prev}, L `// Keep leader`
else
  return v, $\texttt{Selector}(\mathcal{R} \cup \{i\})$ `// Update leader`

When generalizing this approach to cyclic mazes, additional considerations are required as multiple paths may exist between a pair of nodes. The proposed algorithm lets agents send a *directed cast* along a single, chosen edge, modeling directional communication along individual corridors. After an agent i moves from any node v to new node u, it sends a directed cast along edge $\{u, v\}$. This informs its *followers* (i.e., active agents j that have agent i as their leader) about agent i's previous node v. Additionally, the head can send a directed cast to inform any agents in its communication range of its next move.

Similarly to [1], the agents can employ broadcast messages. In the present work, three types of broadcast messages are used:

1. The *status message* is sent once per time step by each agent $i \in \mathcal{A}^{\text{active}}$ and contains its ID, whether i is at the goal, (i.e., $v_i = g$) and i's leader pointer $L_i \in (\mathcal{A}^{\text{active}} \setminus \{i\}) \cup \{\texttt{nil}\}$.
2. The *competing message* is sent once per time step by all active non-head agents and contains only their ID and the flag *competing*, which indicates if they intend to move and should therefore be considered competing in potential conflicts for nodes in this time step.

3. The *head message* is sent once per time step by the current head agent, h, containing its ID, the flag *competing*, the state of the single-agent maze solver and information on the head agent at the next time step L_h (either `nil` if the current head keeps the role, and the ID of another agent otherwise).

The proposed algorithm is shown in Algorithm 1, seen from the point of view of the executing agent i. It is executed by all agents in a synchronous, discrete-time manner. Subscripts i are omitted for better readability.

Initialization: (Line 1) When i first becomes active, it is at node s and has no leader. (Line 2) Agent i then senses the adjacent node (as s is a leaf) to identify the neighborhood as well as if the adjacent node is the goal or if it is occupied. (Line 3) If agent i is the first agent activated, it will not receive any messages, resulting in $\mathcal{C} = \emptyset$. If other agents have been activated before i, then there exists another active agent l that just left the start node and moved to the single adjacent node. In this case, agent i will receive a status message from l, and $\mathcal{C} = l$. (Lines 4–5) If $\mathcal{C} = l$, agent i will make l its leader, $L_i = l$. Otherwise, it keeps the leader pointer $L_i =$ `nil` and assumes the head role.

While Loop: The agent executes the while loop (lines 6–28) until it reaches the goal g. Each iteration of the loop takes one time step. (Line 7) At the beginning of each time step, i resets its target node D to its current node v and competing to `false`. *While Loop Head Decisions:* (Line 8) If $L_i =$ `nil`, then agent i is the head agent. (Lines 9–10) If the goal is adjacent, i sets its target node $D = g$. (Line 12) Else, agent i runs one step of the single-agent maze solver to determine an adjacent potential target node D^{solver}. (Lines 13–14) If D^{solver} is currently occupied by an agent j, j is chosen as the new head and leader of i. (Messages cannot pass through occupied nodes, hence $D^{\text{solver}} \in$ `NodeTowards`(a) will only be fulfilled for agent j, and the `Selector`[8] operator returns j.) Else, D^{solver} is either unoccupied or i's current node, and i retains the head role. (Line 17) If D^{solver} is unoccupied, agent i sends a directed cast along edge v, D^{solver} and updates its target node D accordingly. (Line 18) Then, i broadcasts a head message informing all other agents within its communication range whether one of them has been elected as the new head. *While Loop Follower Decisions:* (Line 20) If at time k agent i has a leader L_i, it executes routine `ResolveConflict` (discussed below), which is shown in Algorithm 2. `ResolveConflict` returns a target node D and the potentially updated leader L_i.

Movement: (Lines 21-22) After completing its decision step, agent i moves synchronously with all other agents to its target node. It updates v^{prev} and v.

Post-Movement Sensing and Messaging: (Lines 23–26) Agent i senses adjacent nodes. If i moved to a new node, it broadcasts a directed cast along $\{v, v^{prev}\}$, and always broadcasts a status message. (Lines 27–28) Agent i receives directed casts and status messages from nearby agents and updates v_L^{prev} and $\mathcal{C}$ accordingly. Agents who terminated after reaching the goal in the previous time step will not send a message, and are removed from $\mathcal{C}$. Lines 23–28 are synchronized with lines 2–5 of a newly activated agent.

[8] The `Selector` operator is any deterministic operator that, given a set of agents $A \subseteq \mathcal{A}$, returns the agent $a \in A$ with $a \leq_{\mathcal{A}} b$ for all $b \in A \setminus \{a\}$.

ResolveConflict: (Line 1) Agent i checks whether the head h is in communication range and not yet at the goal. (Lines 2–3) If the head h is in communication range, agent i waits for the head message. (Lines 4–6) Agent i checks if it has been chosen to be the new head. If so, it broadcasts that it is not competing, sets its leader $L_i = \texttt{nil}$, and sets its target to its current node. (Lines 7–9) If agent i was not chosen as the new head, it checks if the agent h is currently adjacent or will be adjacent in the next step. In this case, it sets the head as its leader, does not compete to move, and sets its target to its current node. If agent i was not chosen as the new head, it checks if it received the directed cast from h or if h is currently in an adjacent node. If either is true, i broadcasts that it is not competing, sets its leader $L_i = h$, and sets its target to its current node. (Lines 10–12) If the current head is not in communication range or neither of the above cases apply, agent i checks if the goal is adjacent and if so, does not compete for other nodes, maintains its current leader L, and sets its target to the goal. (Lines 13–15) Else, if its (non-head) leader is residing in an adjacent node, i keeps this leader and does not move. (Lines 16–18) If none of the above cases apply, agent i competes to move to the node previously occupied by its leader. It broadcasts *competing* = `true` and waits until it receives information on which agents in communication range are competing. (Line 19) Agent i determines which other agents are competing. An agent a is considered competing if it is not yet at the goal node, has signaled that *competing* = `true`, and wants to move to the same node as agent i. (Lines 20–23) The `Selector` function is called on the set of all competing agents and returns a winner. If agent i is the winner, it keeps its previous leader and selects as target the previous node of that leader. Else, i chooses the winner as its new leader and remains at its current node.

Fig. 2 provides an example that highlights particular moments of a group of agents executing the aforementioned algorithms.

3.1 Mathematical Analysis

We now prove the correctness of the algorithm and several properties.

Lemma 1. *At all discrete times $k \in \mathbb{N}_0$, there is exactly one agent assuming the head role. Let $k_w \in \mathbb{N}_0 \cup \{\infty\}$ be the time at which a single agent running a single-agent maze solver resides in a node adjacent to the goal for the first time. At all time steps $k \leq k_w$, using the same single-agent maze solver, the head agent occupies the same node as the single agent would occupy at time k. At all times $k > k_w$, the head agent resides at the goal node.*

Proof. Proof by induction: $k = 0$: At the beginning of the first time step, there is only one active agent h. h resides in the start node $s \neq g$ as would an agent executing the single-agent maze solver and is the only head by default.

$k \to k + 1$: An agent a that activates in time step $k + 1$ will activate only because another agent b just left the start node and moved to an adjacent node. Consequently, a is in communication range of b and will receive a status message from b, choosing b as its leader. At time $k + 1$, a is not the head.

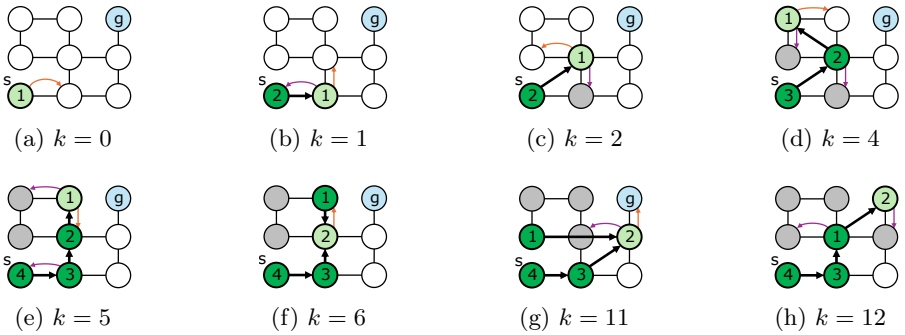


Fig. 2. An example of our algorithm (employing Trémaux's algorithm). Gray nodes have been visited. $k = 0$: All agents are at start node s; agent 1 is active, claiming the head role (light green). $k = 1$: Agent 1 leaves s (now its previous node, purple arrow) following Trémaux's algorithm (orange arrow). Agent 2 becomes active (dark green) and chooses agent 1 as leader (black arrow). $k = 2$–4: Agents 1 and 2 gradually navigate the maze, agent 3 becomes active. $k = 5$: Agent 2 receives the head's directed cast and remains in its current node. Agent 1 chooses agent 2 to become the new head at $k = 6$. $k = 7$–10: Agent 2 transfers the head back to agent 1 which performs backtracking. $k = 11$: Agent 2 is again the head and moves to a node adjacent to the goal. $k = 12$: Agent 2 reaches the goal. Agents 1 and 3 compete for agent 2's previous node; Agent 1 wins and moves to the node. Agent 3 chooses agent 1 as its new leader.

Per the induction assumption, there is exactly one head h at time k that is at the same node as a single agent executing the single-agent maze solver would be. We distinguish between three cases.

Case 1: If $k < k_w$, h will use the single-agent maze solver to determine where to move to. If the determined adjacent node D^{solver} is not occupied, h will keep $L_h = \texttt{nil}$, set $D = D^{\text{solver}}$ and inform agents in communication range (lines 15–18). Any agent that has h in communication range will receive the head message (line 3 of Algorithm 2). As h sent $L_h = \texttt{nil}$, no agent i will become the head (lines 4–6). Moreover, there is no other moment in time where an agent can set its leader to `nil`. As h will move to D^{solver} in time step $k+1$, at time $k+1$, there is exactly one head, agent h, which resides in the same node as a single agent executing the single-agent maze solver would. If D^{solver} is occupied at time k, agent h chooses the occupying agent L as its new leader and sends $L_h = L$ to all agents in communication range via the head message. As L is occupying an adjacent node, it is in communication range of h and will wait for and receive the head message. As $L_h = L$, L sets its leader to `nil` and assumes the head role. As every agent has a unique ID, no other agent $j \in \mathcal{C}_h \setminus \{L\}$ will fulfill $L_h = j$ and assume the head role. Consequently, at time $k+1$, there is exactly one head, agent L, which resides in the same node as the single-agent maze solver would.

Case 2: If $k = k_w$, then h is currently in a node adjacent to the goal. Consequently, it will move to the goal in time step $k + 1$. In doing so, it will remain the head and no other agent will receive a request to assume the head role.

Case 3: If $k > k_w$, then h is at the goal node and no longer sending messages, and will never transfer the head. Hence, no other agent will set its leader to `nil` as $\{h \in \mathcal{C} \mid L_h = \texttt{nil} \wedge v_h \neq g\} = \emptyset$. In both latter cases, at $k+1$, there is exactly one head which resides at the goal.

Lemma 2. *The first visit to any node $v \in \mathcal{V} \setminus \{s\}$ is done by the current head agent. Any agent that reaches the goal will do so via the same node adjacent to the goal that was traversed by the head.*

Proof. In the case where s and g are adjacent, at $k = 0$, there is only one active agent. This agent assumes the head role, moves to g (where it will eventually terminate). As all other agents activates afterwards, they also sense the goal being adjacent, move there and terminate. Consequently, the only node $\neq s$ that is visited is g and it is first visited by the head. All agents move to g via s.

Assume that s and g are not adjacent. Assume that there exists a node $w \in \mathcal{V} \setminus \{s, g\}$ that has not been visited before and is now visited by a non-head agent a. a has only two possible options to move to another node (lines 12 and 21 of Algorithm 2). Since $w \neq g$, a must have moved according to line 21, that is, to a node that was previously visited by its leader L, which is a contradiction to the fact that a is the first agent to visit w. Let $u \in \mathcal{V} \setminus \{s, g\}$ be the first node adjacent to the goal that the head agent visited. As the goal is adjacent, the head moves there from u and terminates. As shown above, no other node adjacent to the goal will be visited by any agent, as the head has already terminated at the goal. Thus, no other agent reaches a node adjacent to the goal before the head agent, and the goal node g is first visited by the head agent. Moreover, as u is the only adjacent node to the goal that is visited by any agent, it directly follows that every agent that reaches the goal will do so by traversing via u.

Lemma 3. *For any node $v \in \mathcal{V} \setminus \{g\}$, at all discrete times k, v is occupied by at most one active agent. If an agent leaves node v in time step k, no other agent that was already active at $k-1$ will move to v in the same time step. In time step k, all agents competing for the same node v have the same leader.*

Proof. Proof by strong induction: At time $k = 0$, all agents are at s, all but one are inactive and the lemma holds for any $v \in \mathcal{V} \setminus \{g\}$. Let $k_0 \in \mathbb{N}$ be the first time step where the head leaves the start node and hence occupies an adjacent node. As soon as the head leaves the start node, one agent a that is not the head becomes active at the start node. Again, the lemma is true.

$1, 2, \ldots, k \to k+1$: Let $v \in \mathcal{V} \setminus \{g\}$. Per induction assumption at k, there is no more than one active agent at v. There are only two options how agents can move to v: either as the (only) head (Lemma 1) moving to $D^{\text{solver}} = v$ or following their leader to $v_L^{prev} = v$. In the first case, if v is currently occupied, the head h chooses the occupying (active) agent (which is only one due to the induction assumption) as new head and does not move (lines 13–14). Hence, h will not move to a node in time step $k+1$ that is occupied at time k. If v is not occupied, h sends out a directed cast along $D^{\text{solver}} = v$ which all agents in $\mathcal{U}_h(v)$ will receive (i.e., all other agents adjacent to v). They choose h as their

leader and remain in their current node. Hence, h can safely move to v without collision, that is, there is only one active agent at v at time $k+1$.

In the second case, an agent a aims to move to the previous node $v_{L_a}^{prev} = v$ of its leader L_a. It can only do so if it received the directed cast from its leader L_a in the previous time step, that is, $a \in \mathcal{U}_{L_a}(v)$ and $L_a \in \mathcal{C}_a$, if it is selected by the `Selector`-operator, and if the head does not aim to move there in this time step. If $v_{L_a}^{prev}$ is occupied, L_a must be residing there and a does not move (lines 13–15) as else, a would not have received the directed cast from L_a. If $v_{L_a}^{prev}$ is unoccupied, a receives all competing messages from agents in communication range. Every non-head agent b that could move to $v_{L_a}^{prev}$ fulfills the following: (i) $b \in \mathcal{C}_a$, as b must be adjacent to $v_{L_a}^{prev}$ to be able to move there, so a and b are two hops away with an empty node in between, (ii) $v_b \neq g$ as else, b would not move, (iii) $competing_b = \texttt{true}$, as this is only sent if b not already decided to remain in its current node nor move to g, (iv) $L_b \in \mathcal{C}_a$ and $v_{L_b}^{prev} = v_{L_a}^{prev}$, as else, b would not aim to move to $v_{L_a}^{prev}$. Per induction assumption at $k-1$, there was at most one agent residing at $v_{L_a}^{prev}$, which means that in time step $k+1$, $v_{L_a}^{prev}$ can only be the previous node of L_a and no other agent. Hence, every non-head agent that aims to move to and therefore competes for $v_{L_a}^{prev}$ in time step $k+1$ does so because L_a is its leader. Consequently, a has every competing agent in $\mathcal{R}_a$ (line 19). It directly follows that $b \in \mathcal{R}_a \iff a \in \mathcal{R}_b$ and $\mathcal{R}_a \cup \{a\} = \mathcal{R}_b \cup \{b\}$. All agents competing for v choose the same agent as winner by applying the (deterministic) `Selector`-operator on the same set, resulting in only one agent that is allowed to move to v.

Hence, at time $k+1$, there is at most one active agent at v. As agents only move simultaneously and to currently unoccupied nodes, no other agent that is active at k will aim to move to v in time step $k+1$ if v is occupied at k.

Corollary 1. *Under the algorithm, vertex and following conflicts cannot occur.*

Lemma 4. *At all discrete times k, if an active agent a is not the head, it has its current leader L_a in communication range and one or more of the following three cases are true:*

1. *the previous node of its leader $v_{L_a}^{prev}[k]$ is unoccupied and adjacent to $v_a[k]$ and $v_{L_a}[k]$*
2. *the current node of a's leader $v_{L_a}[k]$ is adjacent to a's current node $v_a[k]$*
3. *$v_a[k] = v_{L_a}[k] = g$.*

Proof. See the supplementary material for the complete proof. Non-head agents move to follow their leader. Agents only compete to move when their leader has moved to a non-adjacent node. An activated agent starts with their leader in an adjacent node by default, so this occurs only when a leader moves two nodes away, leaving an unoccupied node. The leader is still in communication range at this point, but may move again. Agents following this leader now compete for the unoccupied space. If an agent wins this competition, it moves to follow its leader. Otherwise, another agent moves to this space and is chosen as new leader, maintaining proximity and communication network cohesiveness. Leaders

may also stop communicating when they reach a goal. In this case, agents still compete for the previously occupied node, which brings them into proximity of the goal and maintains the network connectivity for remaining agents.

Lemma 5. *At all discrete times k, an active agent follows the head directly or indirectly. Formally, at discrete time k, let the directed leadership graph be $\mathcal{T}_L[k] = (\mathcal{A}^{active}[k], \mathcal{E}_L[k])$, $a, b \in \mathcal{A}^{active}[k] : b$ is leader of $a \iff (a, b) \in \mathcal{E}_L[k]$. Then, $\mathcal{T}_L[k]$ is an* in-tree *[14] with the current head agent h as its root, that is, for all $a \in \mathcal{A}^{active}[k] \setminus \{h\}$, there exists exactly one path to h in $\mathcal{T}_L[k]$.*

Proof. See the supplementary material for the complete proof. For an agent to not be following the head (either directly or indirectly), it would have to be following another agent for which no path exists leading to the head. This would require either a cycle within the tree created by following agents, or for two agents to be acting as the head. Lemma 1 shows that only one head exists at a time. A cycle in leadership can only arise when the head passes the head role to a new agent who is choosing to follow the current head. However, once an agent receives a head message informing it of its role as new head, it removes the following edge to the former head. Additionally, when agents win competitions for moving to unoccupied nodes, they maintain their leader and losing agents select the winner as their new leader, maintaining the in-tree behavior of leadership.

Theorem 1. *If a single agent executing a single-agent maze solver reaches a node adjacent to the goal for the first time in time step $k_g - 1$, then n agents executing Algorithm 1 (using the same single-agent maze solver) reach the goal in at most $k_g + 2(n-1)$ time steps.*

Proof. If the goal node is adjacent to the start node, the single agent is already in an adjacent node at time $k = 0$, that is, $k_g = 1$. Executing Algorithm 1, the first agent moves to g in the first time step (lines 9–10), that is, it will be there at $k = 1$. The newly activated agent moves to the goal in the next time step (Algorithm 2, lines 10–12), and this repeats for all other agents, that is, all agents will reach the goal at time $n \leq k_g + 2(n-1)$.

Now, let s and g be not adjacent. Let $w \neq s$ be the first node adjacent to the goal visited by the single agent in time step $k_g - 1$. Per Lemma 1, the current head agent h is in node w at time $k_g - 1$, moves to the goal node in the next time step, $v_h[k_g] = g$, and send out a final directed cast along w and status message (including that it is now at the goal node) before it terminates. This means that no other agent receives the head role after k_g.

Let k_a be the time step in which an agent a moves to the goal node. Let $n_a - 1$ be the number of agents that are not yet at the goal node at time k_a. We prove the following statement via induction: *If an agent a reaches the goal at k_a, then all $n_a \leq n$ agents reach the goal in at most $k_a + 2(n_a - 1)$ time steps.*

$n_a = 1$: If a reaches the goal at k_a, there are $n_a - 1 = 0$ agents left on nodes other than the goal. All agents reached the goal at $k_a = k_a + 2(n_a - 1)$.
$n_a \to n_a + 1$: If a moved to the goal at k_a, there are $(n_a + 1) - 1 = n_a > 0$ agents not at g. None of these agents can assume the head role (Lemmas 1 and

2). Assume that there is an agent b of these n_a agents that is not a direct or indirect follower of a. As b is a direct or indirect follower of the head (Lemma 5), there exists exactly one path in the directed leadership graph $\mathcal{T}_L[k_a]$ from b to the head h. Let the set $\mathcal{B}$ include all agents that are between b and h on this path (especially, $a \notin \mathcal{B}$). If $\mathcal{B} = \emptyset$, b is a direct follower of h. Let $c \in \mathcal{B} \cup \{b\}$ be the agent on this path that is the direct follower of h. h has already been at the goal node for at least two time steps (i.e., $v_{prev_h} = g$), as it must have moved there before agent a moved to w at time $k_a - 1$. Hence, c must also be at the goal node (Lemma 4), as it cannot be adjacent to g, as w was occupied by a at $k_a - 1$ (Lemmas 2 and 3). With the same argument, c must have been at the goal node since at least $k_a - 2$, and if $c \neq b$, we can follow that the direct follower of c in $\mathcal{B}$ is already at the goal. Continuing this down the path to b, b must already be at the goal node at time k_a. Consequently, all n_a agents that are not yet at the goal node are direct or indirect followers of a.

This means that at time k_a, there exists at least one direct follower b that is in a node adjacent to w. In the next time step $k_a + 1$, b aims to move to $v_{prev_a} = w$. Let $\hat{b}$ be the agent that wins against all competing agents for w, keeps a as leader (Lemma 3) and moves there at $k_a + 1$. Then, in the next time step $k_a + 2$, as the goal is adjacent, $\hat{b}$ will move to g and consequently be there at time $k_a + 2$ while $n_a - 1$ agents are not at the goal at $k_a + 2$. Using the induction assumption, it follows that all agents reach the goal in at most $(k_a + 2) + 2(n_a - 1) = k_a + 2((n_a + 1) - 1)$ time steps.

Hence, as the head agent reaches g at time k_g, it follows that all n agents reaches the goal in at most $k_g + 2(n - 1)$ time steps.

Theorem 2. *Let the full-knowledge (FK) strategy be the strategy where agents have full prior knowledge of the environment. If the used single-agent maze solver is complete, the ratio $R(n) = M_{Algo}(n)/M_{FK}(n)$ of the makespan of the proposed algorithm, $M_{Algo}(n)$, relative to the one of the FK strategy, $M_{FK}(n)$, is either $R(n) = 1$ or strictly decreasing with asymptotic equivalence, $\lim_{n\to\infty} R(n) = 1$.*

Proof. If the used single-agent maze solver is complete, it finds a node adjacent to the goal at time $k_g - 1$. Per Theorem 1, n agents executing Algorithm 1 reach the goal at time $M_{Algo}(n) = k_g + 2(n - 1)$ at the latest.

If start and goal are adjacent, using Algorithm 1, at every time step, one agent reaches the goal, and all agents will have reached it at time $M_{Algo}(n) = n$. Having full knowledge of the environment, agents executing the full-knowledge strategy also move one-by-one to the goal, such that all agents will have reached the goal at time $M_{FK}(n) = n$. Hence, $R(n) = M_{Algo}(n)/M_{FK}(n) = 1$.

Now assume that s and g are not adjacent. As s is a leaf and the full-knowledge strategy needs to have no vertex or following conflicts at nodes $\neq g$, an agent can leave the start node only every two time steps. This means that, even when following shortest paths, an agent can move to g only every two time steps. Consequently, $M_{FK}(n) = k_{FK} + 2(n-1)$, where k_{FK} describes the length of the shortest path from s to g, as the first agent that left s reaches g at k_{FK} and every two time steps later, the next agent can reach the goal. If $k_g = k_{FK}$,

$R(n) = 1$. If $k_g > k_{FK}$, it follows $(\partial_n R)(n) = -2\frac{k_g - k_{FK}}{(k_{FK}+2(n-1))^2} < 0$ and

$$\lim_{n\to\infty} R(n) = \lim_{n\to\infty} \frac{k_g + 2(n-1)}{k_{FK} + 2(n-1)} = \lim_{n\to\infty} \frac{k_g/(n-1) + 2}{k_{FK}/(n-1) + 2} = 1.$$

Hence, the makespan of Algorithm 1 is either equal or asymptotically equivalent to the one of the full-knowledge strategy with respect to n.

3.2 Time and Space Complexity Analysis

Let $d \coloneqq \deg \mathcal{G}$ be the maximum degree of the graph.

Space Complexity. In the worst case, an agent a is at a node with degree d, and all adjacent nodes are unoccupied, have degree d and have active agents occupying adjacent nodes. This results in at most $\min(n, d(d-1))$ active agents in communication range of a, and requires $\mathcal{O}(\min(n, d^2))$ space to store $\mathcal{C}$ and information sent by nearby agents. Agents must also maintain the `NodesTowards` list for each agent within communication range. In the worst case, an agent in a non-adjacent node sends a broadcast that reaches a via all d adjacent nodes, resulting in a worst-case size of $\mathcal{O}(d)$ for any `NodesTowards`. Agents also require $\mathcal{O}(d)$ space for $\mathcal{N}^{\text{occupied}}$, and $\mathcal{O}(1)$ for any other variables. Our proposed algorithm therefore requires $\mathcal{O}(d \min(n, d^2))$ space. Additional spatial requirements depend on the single-agent maze solver selected (e.g., BFS has a requirement of $\mathcal{O}(|\mathcal{V}|)$ [8], resulting in $\mathcal{O}(|\mathcal{V}| + d \min(n, d^2))$, whereas Trémaux's algorithm without marking the environment requires $\mathcal{O}(|\mathcal{E}|)$ [8], giving a total of $\mathcal{O}(|\mathcal{E}| + d \min(n, d^2)))$.

Time Complexity. Each iteration of the algorithm has a constant number of function calls that have at maximum a runtime of $\mathcal{O}(\min(n, d^2))$. Per Theorem 1, the overall number of iterations depends on the single-agent maze solver and number of agents. Trémaux's algorithm takes $\mathcal{O}(|\mathcal{E}|)$ as all edges are traversed at most twice [8], BFS needs $\mathcal{O}((|\mathcal{V}| + |\mathcal{E}|)^2)$ [8] as a physical agent needs to backtrack at every visited node, and a uniform random walk is expected to run in $\mathcal{O}(|\mathcal{V}|^3)$ [3]. In total, the algorithm with Trémaux's algorithm needs $\mathcal{O}((|\mathcal{E}| + n) \min(n, d^2))$, with BFS $\mathcal{O}(((|\mathcal{V}| + |\mathcal{E}|)^2 + n) \min(n, d^2))$ and with a uniform random walk an expected $\mathcal{O}((|\mathcal{V}|^3 + n) \min(n, d^2))$.

4 Results

Our algorithm (hereafter, MAMT) requires a single-agent maze solver to determine head movement. We evaluated MAMT with three such solvers: Trémaux's algorithm, which explores the graph in a DFS-manner, ensuring no edge is visited more than twice; BFS, which explores the graph with increasing distance from the start node; and a uniform random walk. Simulations were conducted in square grid mazes (see Fig. 3). For every combination of maze size $l \times l, l \in \{5, 15, 25, 35\}$ and number of agents $n \in \{1, 5, 25, 125, 625\}$, a same set of 20 random mazes were simulated, with a 10^4 steps timeout. Fig. 4 shows the makespan and sum-of-fuels performance. All trials were collision-free, and, for

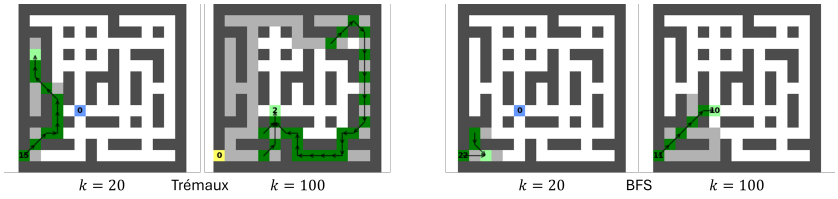


Fig. 3. Simulations in grid mazes using Trémaux's algorithm and BFS at times $k = 20$ and $k = 100$. Walls are dark gray, and nodes that were already visited are light gray. In this example, Trémaux's algorithm explores a larger portion of the graph, though BFS is the first to reach the goal.

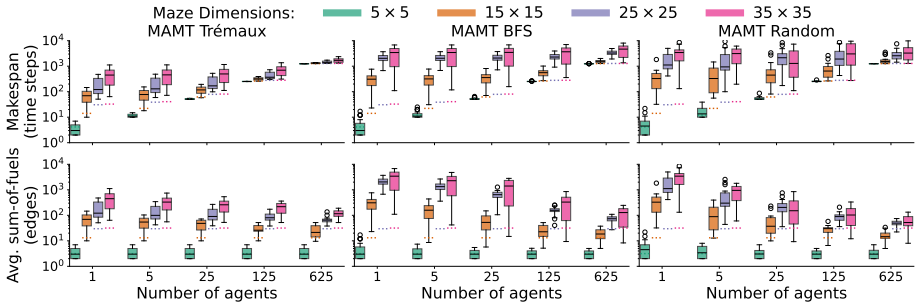


Fig. 4. Makespan and average sum-of-fuel for the proposed algorithm using Trémaux's algorithm, BFS, and uniform random walk across various mazes and number of agents. Dashed lines indicate the mean values for agents following the shortest path.

MAMT with Trémaux's algorithm or BFS, all agents reached the goal. In ten trials (two for $l = 25$ and eight for $l = 35$), MAMT with random walk timed out. Although in contrast to BFS, Trémaux's algorithm does not necessarily find a shortest path, its performance was superior to that of BFS. This result arises because Trémaux's algorithm backtracks less extensively than BFS. As the group size increases, both makespan and average sum-of-fuels showed a tendency of approaching the mean performance of a full-knowledge strategy (i.e., omniscient agents, following a shortest path). Once the goal is found by the first agent, further agents follow at a steady rate, reducing average sum-of-fuel and the increase in makespan as the number of agents grows .

We evaluate MAMT against the full-knowledge strategy and a naïve baseline. For the latter, every agent a independently runs the single agent maze solver to determine its next node x_a. The naïve baseline uses the following rules: (i) If x_a is currently unoccupied and no other agents want to move there, a moves to this node; (ii) if x_a is currently unoccupied and at least one other agent wants to move there, the agent of minimum ID moves there while other competing agents wait; (iii) A *target-cycle* occurs if it exists a subset of agents $\mathcal{B} \subseteq \mathcal{A}$ and a permutation $\pi : \mathcal{B} \mapsto \mathcal{B}$ with $x_b = v_{\pi(b)}\ \forall b \in \mathcal{B}$. If such a target-cycle occurs and a is part of

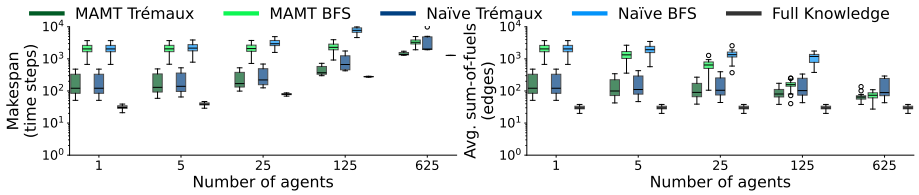


Fig. 5. Comparison of makespan and average sum-of-fuel for MAMT and naïve strategies (each using the Trémaux's algorithm and BFS) against the full-knowledge strategy. Trials were conducted for up to 625 agents in 20 mazes of size 25×25. Trials that triggered the timeout of 10^4 time steps were excluded.

it, it forwards its single-agent maze solver instance to the agent that occupies its target node (and receives an instance of another agent); (iv) if x_a is currently occupied and a is not part of a target-cycle, it waits. Trials were performed in mazes of size 25×25 with the remaining setup unchanged. The results are shown in Fig. 5. The MAMT variants succeeded in all trials. For 125 agents, the naïve strategy using BFS timed out in 9 trials. For 625 agents, the naïve strategy with Trémaux's algorithm timed out in 2 trials, whereas the BFS-based variant timed out in all 20 trials . As before, the superior performance between Trémaux's algorithm compared to BFS is highly visible. The results show that with growing agent group size and the same underlying single-agent maze-solver, MAMT outperforms the naïve strategy in both makespan and average sum-of-fuels. They suggest that MAMT approaches in makespan the full-knowledge strategy as number of agents increases. The source code is available at [21].

5 Conclusion

We presented an algorithm for multi-agent maze traversal, extending [1] to general graphs. We prove that if the underlying single-agent maze solver is complete, the algorithm enables all agents to reach the undisclosed goal in finite time. We prove that the achieved makespan is asymptotically equivalent to the one achieved by optimal, omniscient agents. Simulations show that Trémaux's algorithm outperforms a BFS and random solver and that the algorithm outperforms a naïve baseline where all agents use the single-agent maze solver. Future work includes real-world experiments [1] and robustness to communication failures.

Acknowledgments. This work was co-funded by EU Horizon Europe Framework Programme ("OpenSwarm"; grant 101093046); BMBF (Robotics Institute Germany; grant 16ME1001); LOEWE center emergenCITY [LOEWE/1/12/519/03/05.001(0016)/72]. ChatGPT (OpenAI, Free Plan) assisted generation of code used for generating maze environments for simulation purposes, and code to plot Figures 3–5. It was also used for improving the language of the manuscript. It was not used in the design or implementation of the presented algorithm, baseline methods, or proofs.

Disclosure of Interests. The authors have no competing interests to declare.

References

1. Argote-Gerald, J., Miyauchi, G., Rau, J., Trodden, P., Groß, R.: Design for one, deploy for many: Navigating tree mazes with multiple agents. In: 2025 IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS). pp. 1–7 (2025). https://doi.org/10.1109/MRS66243.2025.11357261
2. Brass, P., Cabrera-Mora, F., Gasparri, A., Xiao, J.: Multirobot tree and graph exploration. IEEE Transactions on Robotics **27**(4), 707–717 (2011). https://doi.org/10.1109/TRO.2011.2121170
3. Brightwell, G., Winkler, P.: Maximum hitting time for random walks on graphs. Random Structures & Algorithms **1**(3), 263–276 (1990). https://doi.org/10.1002/rsa.3240010303
4. Cabrera-Mora, F., Xiao, J.: A flooding algorithm for multirobot exploration. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **42**(3), 850–863 (2012)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2022)
6. Crnković, B., Ivić, S., Zovko, M.: Fast algorithm for centralized multi-agent maze exploration. arXiv preprint arXiv:2310.02121 (2023)
7. Dereniowski, D., Disser, Y., Kosowski, A., Pajak, D., Uznański, P.: Fast collaborative graph exploration. Information and Computation **243**, 37–49 (2015)
8. Even, S.: Graph Algorithms. Cambridge University Press, 2nd edn. (2011)
9. Fraigniaud, P., Gasieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks: An International Journal **48**(3), 166–177 (2006)
10. Hall, M.D., Özdemir, A., Groß, R.: Self-reconfiguration in two-dimensions via active subtraction with modular robots. In: Robotics: Science and Systems (2020)
11. Kivelevitch, E.H., Cohen, K.: Multi-agent maze exploration. Journal of Aerospace Computing, Information, and Communication **7**(12), 391–405 (2010)
12. Linardakis, M., Varlamis, I., Papadopoulos, G.T.: Distributed maze exploration using multiple agents and optimal goal assignment. IEEE Access **12**, 101407–101418 (2024). https://doi.org/10.1109/ACCESS.2024.3431909
13. Martz, J., Al-Sabban, W., Smith, R.N.: Survey of unmanned subterranean exploration, navigation, and localisation. IET Cyber-Systems and Robotics **2**(1), 1–13 (2020)
14. Mehlhorn, K., Sanders, P.: Algorithms and Data Structures: The Basic Toolbox. Springer Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77978-0
15. Murphy, R.R., Kravitz, J., Stover, S.L., Shoureshi, R.: Mobile robots in mine rescue and recovery. IEEE Robotics & Automation Magazine **16**(2), 91–103 (2009)
16. Nagavarapu, S.C., Vachhani, L., Sinha, A.: Multi-robot graph exploration and map building with collision avoidance: A decentralized approach. Journal of Intelligent & Robotic Systems **83**(3), 503–523 (2016)
17. Nebel, B., Bolander, T., Engesser, T., Mattmüller, R.: Implicitly coordinated multi-agent path finding under destination uncertainty: Success guarantees and computational complexity. Journal of Artificial Intelligence Research **64**, 497–527 (2019)
18. Parrott, C., Dodd, T.J., Boxall, J., Horoshenkov, K.: Simulation of the behavior of biologically-inspired swarm robots for the autonomous inspection of buried pipes. Tunnelling and Underground Space Technology **101**, 103356 (2020)
19. Petráček, P., Krátký, V., Petrlík, M., Báča, T., Kratochvíl, R., Saska, M.: Large-scale exploration of cave environments by unmanned aerial vehicles. IEEE Robotics and Automation Letters **6**(4), 7596–7603 (2021). https://doi.org/10.1109/LRA.2021.3098304

20. Queffelec, A., Sankur, O., Schwarzentruber, F.: Complexity of planning for connected agents in a partially known environment. Theoretical Computer Science **941**, 202–220 (2023)
21. Rau, J., Argote-Gerald, J., McFassel, G., Miyauchi, G., Trodden, P., Groß, R.: Simulation source code. https://git.rwth-aachen.de/julian.rau/multi-agent-maze-traversal-on-cyclic-graphs (2026)
22. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artificial intelligence **219**, 40–66 (2015)
23. Shofer, B., Shani, G., Stern, R.: Multi agent path finding under obstacle uncertainty. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 33, pp. 402–410 (2023)
24. Silver, D.: Cooperative pathfinding. In: Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment. vol. 1, pp. 117–122 (2005)
25. Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T.T., Li, J., Atzmon, D., Cohen, L., Kumar, T.K.S., Boyarski, E., Barták, R.: Multi-agent pathfinding: Definitions, variants, and benchmarks. In: Proceedings of the Symposium on Combinatorial Search (SoCS). vol. 10, pp. 151–158 (2019). https://doi.org/10.1609/socs.v10i1.18510
26. Yu, J., LaValle, S.M.: Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. IEEE Transactions on Robotics **32**(5), 1163–1177 (2016). https://doi.org/10.1109/TRO.2016.2593448
27. Zhang, J., Niu, X., Croxford, A.J., Drinkwater, B.W.: Pipe inspection using guided acoustic wave sensors integrated with mobile robots. NDT & E International **139**, 102929 (2023). https://doi.org/https://doi.org/10.1016/j.ndteint.2023.102929, https://www.sciencedirect.com/science/article/pii/S0963869523001445

Proofs of Lemmas 4–5

In the following, the full proofs of Lemmas 4–5 are presented.

5.1 Proof of Lemma 4

Proof. Proof by induction. At $k = 0$, there is only one active agent. It assumes the head role and the lemma is true. Let k_0 be the first discrete time where the head is not in the start node anymore, meaning it left the start node in time step k_0 and is consequently in an adjacent node. As soon as the head left the start node, a new agent a that is not the head became active. As this agent then received a status message from the head, a chose the head as its leader. As the head is adjacent, a has its leader in communication range.

$k \to k+1$: If a was just activated at the start node in time step $k+1$, it has L_a in communication range (same argument as above) and $v_{L_a}[k+1]$ is adjacent. Let h be the head agent at time k. If it remains the head in time step $k+1$, it has no leader at $k+1$. Otherwise, there was another active agent L that occupied an adjacent node of h at k. L was then chosen to be the new head and h's leader. As h did not leave its current node, $v_h[k] = v_h[k+1]$. In the same time step, L received the head message stating that L should assume the head role, and consequently did, remaining in its current node, $v_L[k] = v_L[k+1]$. Hence, L is adjacent to h and in h's communication range at time $k+1$.

Let a be an active agent that does not assume the head role in time step $k+1$ and was activated in an earlier time step. Per induction assumption, a had its leader L_a in communication range and at least one of the three cases is true at k. If $v_a = v_L = g$, neither a nor L will move in time step $k+1$ and the lemma is true for $k+1$.

Let $v_a \neq g$ at k. As a is not the head, it will execute Algorithm 2. If the head h is not yet at the goal and in communication range of a, a will wait for the head message from h. As a will not assume the head role, it will check if it received the head's directed cast or if the head is already in an adjacent node and not a's current leader. If so, a chooses h as its new leader and remains in its current node. If the directed cast was received by a, D^{solver} is an adjacent node of v_a where the head will move to in this time step. If the head is already in an adjacent node $v_h[k]$, it can only move to a node that is at most two edges away from a. In this case, no other agent will move to $v_h[k]$ in time step $k+1$ (Lemma 3). Hence, a's new leader h is adjacent to a or two nodes away with the unoccupied node $v_{prev_h}[k+1] = v_h[k]$ adjacent to $v_a[k+1]$ and $v_h[k+1]$, meaning that $h \in \mathcal{C}_a$ at time $k+1$. If the head is not in communication range or already at the goal, or no directed cast was received and (i) the head is not adjacent or (ii) already a's leader, a checks if the goal is adjacent. If so, the head has been at the goal for multiple time steps (Lemmas 2 and 3) and a moves there in timestep $k+1$. As a must have moved to this adjacent node of the goal in the time step k (else it would have already moved to the goal in a previous time step), a followed its leader L_a there, meaning that L_a already resides at the goal, i.e., $v_a[k+1] = v_{L_a}[k+1] = g$. If the goal is not adjacent to a, a checks if

L_a resides in an adjacent node at time k. If so, a will keep L_a as leader and not move. Either L_a also remains at its current node (i.e., at time $k+1$ a is adjacent to its leader L_a) or it moves to an adjacent node and no other agent moves to $v_{L_a}[k]$ (Lemma 3) meaning that L_a is in communication range and its previous node $v_L^{prev}[k+1] = v_L[k]$ is unoccupied and adjacent to a and L_a at time $k+1$.

If none of the above applies (i.e., at time k the leader is two nodes away from a and the leader's previous node $v_{L_a}^{prev}[k]$ is unoccupied and adjacent to $v_a[k]$ and $v_{L_a}[k]$), a calculates all competing agents for $v_{L_a}^{prev}[k]$ (Lemma 3). If a is chosen as the winner, it keeps its leader and moves to $v_{L_a}^{prev}[k]$. As the leader remains in its current node or moves to an adjacent node, at time $k+1$, either L_a is adjacent to a or two nodes away with $v_{L_a}^{prev}[k+1] = v_{L_a}[k]$ which is adjacent to a (as $v_a[k+1] = v_{L_a}^{prev}[k]$) and no other agent moves to $v_{L_a}[k]$ (Lemma 3), i.e., L is in communication range. If a is not selected as winner, it remains in its current node and choses the winning agent b (which moves to $v_{L_a}^{prev}[k]$) as new leader. hence, at time $k+1$, a is adjacent to its new leader b and b is in a's communication range.

5.2 Proof of Lemma 5

Proof. Per Lemma 1, at all discrete times k only one agent has no leader. Every other agent has one leader, meaning that $\left|\mathcal{E}_L[k]\right| = \left|\mathcal{A}^{\text{active}}[k]\right| - 1$. Hence, if $\mathcal{T}_L[k]$ is connected, the underlying undirected graph must be a tree [14], i.e. there is never more than one path from any other node to the root node.

Proof by induction. At $k = 0$, only one agent is active and assumes the head role, the lemma is true. Let k_0 be as in Lemma 4; after the head leaves s, the newly activated agent chooses the head as its leader. As the head moved to an adjacent node of s in time step k_0, it will not choose the newly activated agent as leader (Lemma 1) and the lemma is true.

$k \to k+1$: Assume that at time k, $\mathcal{T}_L[k]$ is an in-tree. There are four cases where an agent can change its leader in time step $k+1$. We interpret every case as operation on $\mathcal{T}_L[k]$ and show that none change the in-tree property: (i) The head chooses another agent b to be the new head and its leader (Algorithm 1 line 14). (ii) A non-head agent a chooses the current head as its new leader (Algorithm 2 line 7). (iii) A non-head agent a competes with other non-head agents for the previous node of its leader and chooses the winner b as new leader (Algorithm 2 line 23). (iv) An inactive agent a activates at the start node s and chooses the agent b that just left s as its leader.

Cases (i) and (ii) can happen simultaneously at some time $k+\epsilon$, $\epsilon \in (0,1)$. In case (i), the edge (h,b) was added to $\mathcal{E}_L$ before h sent out the head-message. b will set its leader to `nil` at $k+\epsilon$ after receiving the head-message, removing the edge (b, L_b) from b to its old leader L_b. Hence, b has no outgoing edges anymore and becomes the new root. For an agent $a \in \mathcal{A}^{\text{active}}[k] \setminus \{b,h\}$ that was a direct or indirect follower from b at time k, i.e., the unique path to h was $P(a,h)[k] = P(a,b)[k] \cup \{(b,L_b)\} \cup P(L_b,h)[k]$, $P(a,b)[k] = P(a,b)[k+\epsilon]$ was not changed and is now the unique path to the new head b. For h, the unique path to the new head b is defined by $P(h,b)[k+\epsilon] = \{(h,b)\}$. For an agent $a \neq b,h$

that was a descendant [14] from h but not b at time k, i.e., $(b, L_b) \notin P(a,h)[k]$, the unique path to the new head b results from $P(a,b)[k] = P(a,h) \cup \{(h,b)\}$. In case (ii), the edge from a to its old leader is removed and the edge (a,h) is added. If h keeps the head role, the unique path from a to h evaluates to $P(a,h)[k+\epsilon] = \{(a,h)\}$. If b is chosen as new head, the path to b results to $P(a,b)[k+\epsilon] = \{(a,h),(h,b)\}$ (see (i)). Hence, $\mathcal{T}_L[k+\epsilon]$ is an in-tree.

At a later time $k+\delta$, $\delta \in (\epsilon, 1)$ case (iii) may occur. In case (iii), as b will keep its current leader in time step $k+1$ (Algorithm 2 line 21, Lemma 3), the unique path to the current head h results from $P(a,h)[k+\delta] = \{(a,b)\} \cup P(b,h)[k+\delta]$. Even if the same update occurs along the leader chain of b, with the same argumentation there will be a unique path $P(b,h)[k+\delta]$ from b to h, i.e., $\mathcal{T}_L[k+\delta]$ is an in-tree.

After moving, at time $k+\gamma$, $\gamma \in (\delta, 1)$ case (iv) may occur and $\mathcal{A}^{\text{active}}[k+\gamma] = \mathcal{A}^{\text{active}}[k+\delta] \cup \{a\}$. As agent a chooses b as its leader, i.e., adding edge (a,b), the unique path to the current head h is defined by $P(a,h)[k+\gamma] = \{(a,b)\} \cup P(b,h)[k+\gamma]$. As no other leadership changes happen, at $k+1$, $\mathcal{T}_L[k+1] = \mathcal{T}_L[k+\gamma]$ is an in-tree.